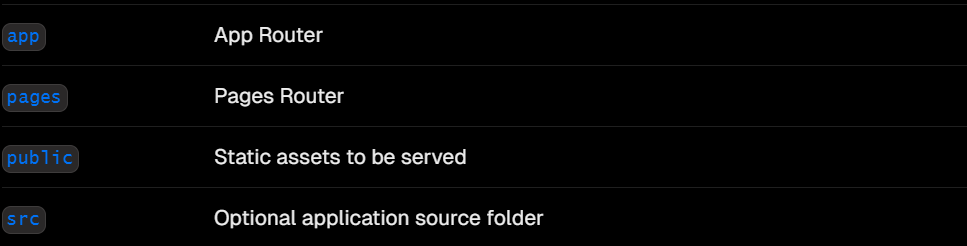
**NEXT.JS  
---------------------------------------------------------------**

🡪 Next JS, is fast, reliable, and Search engine friendly **framework** application. Has it’s own Router, CLI, nodejs server, and compiler. Hence, we can achieve Full stack development using Next.js.  
🡪 Next JS allows us to do ***SSR****(****Server Side Rendering***) by running React code in server and generate HTML and pass it to the client side to render, which leads to the fast and search engine friendly application.

🡪 With Next.js we can pre-render certain pages and components that have static data, when we build our application [ render once server multiple times] using (npm run build).  
This technique is called ***SSR*** *(* ***Static Site Generation*** *)*

**FOLDER STRUCTURE**:-  
It covers top-level **files and folders, configuration files, and routing conventions within the app and pages directories.** All component under APP folder are Server Components**.**  
 Top-level files are used to configure your application, manage dependencies, run middleware, integrate monitoring tools, and define environment variables.  
*So in Next JS, files and folders converted to ROUTES (* routing is based on the file/folder structure). Any other folder are not accessible publicly until PAGE.tsx is mentioned.

  
next.config.js Configuration file for Next.js

package.json Project dependencies and scripts

instrumentation.ts OpenTelemetry and Instrumentation file

middleware.ts Next.js request middleware

.env Environment variables

.env.local Local environment variables

.env.production Production environment variables

.env.development Development environment variables

.eslintrc.json Configuration file for ESLint

.gitignore Git files and folders to ignore

next-env.d.ts TypeScript declaration file for Next.js

tsconfig.json Configuration file for TypeScript

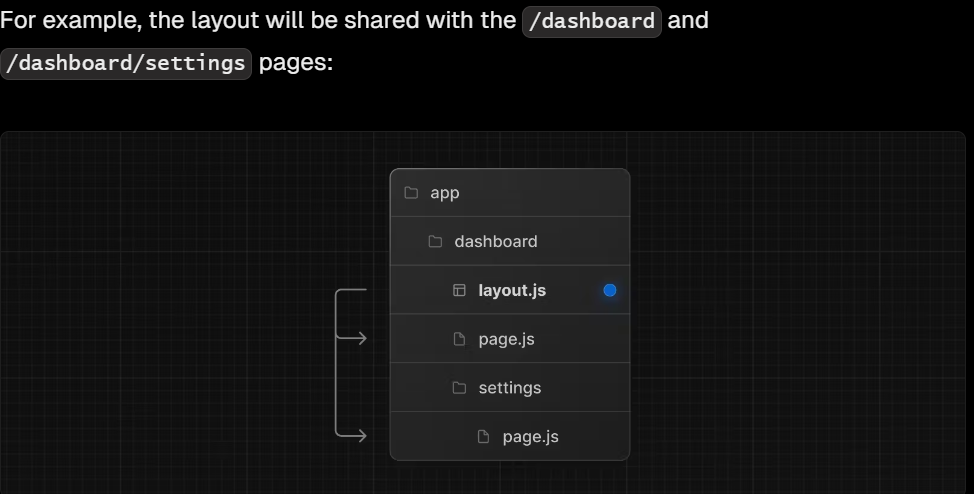
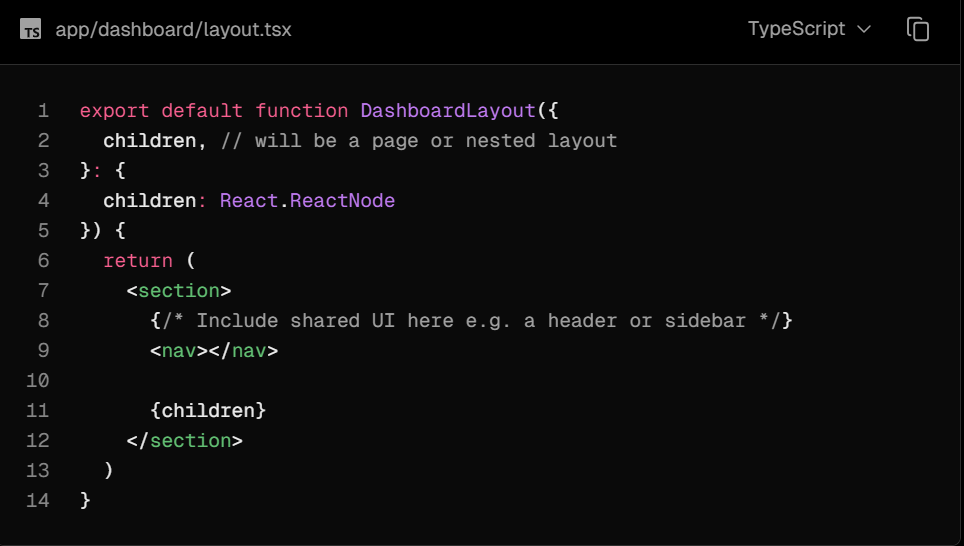
jsconfig.json Configuration file for JavaScript

**Layouts:-**

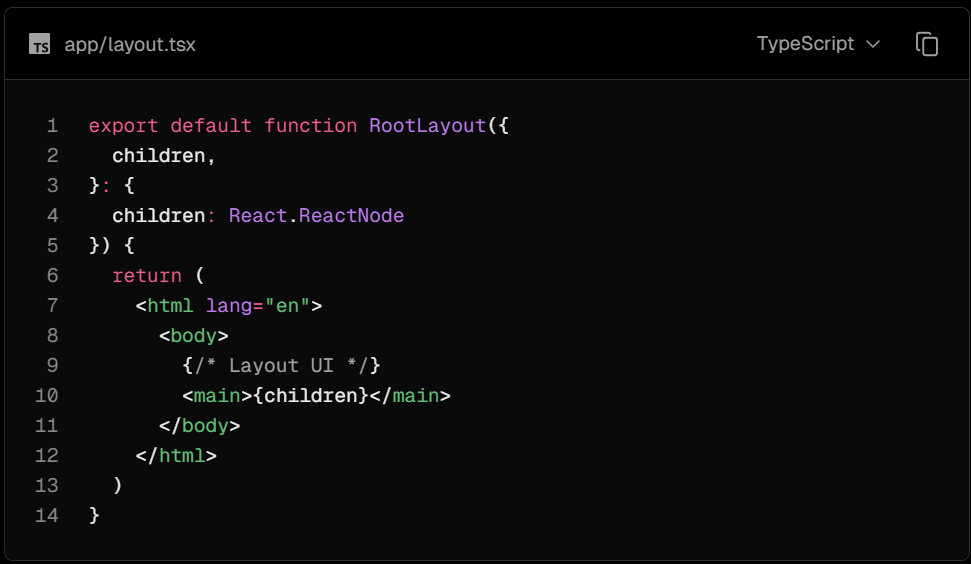
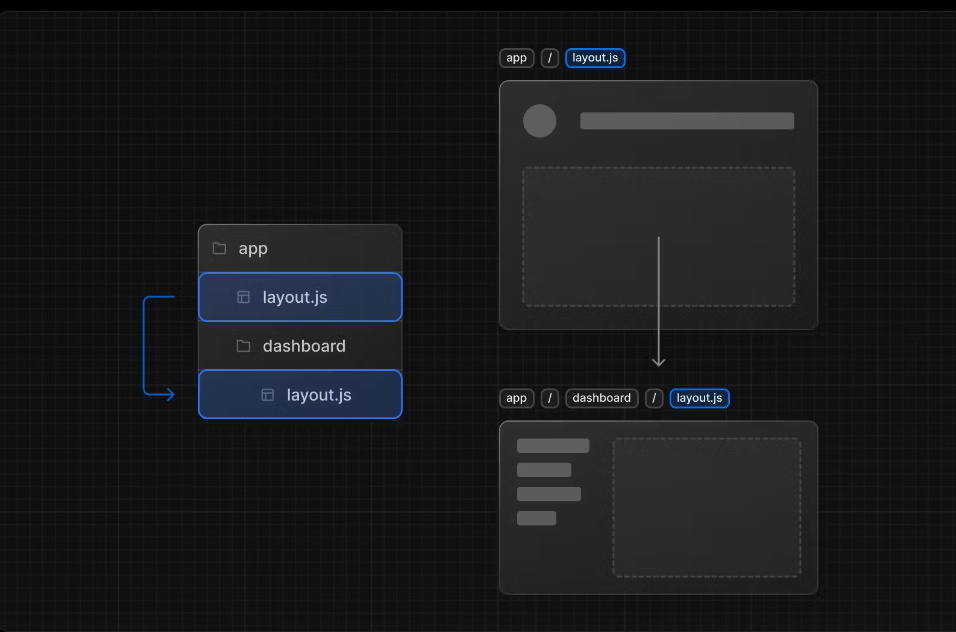
A layout is **UI that is shared between multiple routes**. On navigation, layouts preserve state, remain interactive, and **do not re-render**. Layouts can also be nested.

You can define a layout by default exporting a React component from a layout.js file. The component **should accept a children prop** that will be populated with a child layout (if it exists) or a page during rendering.

**Layouts are Server Components by default** but can be set to a Client Component.

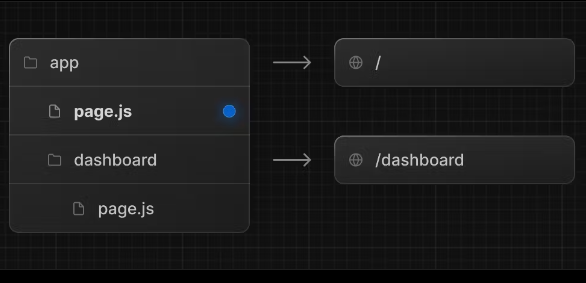
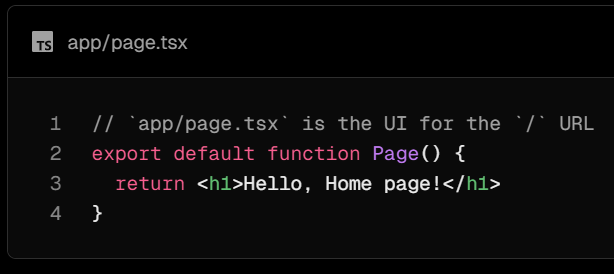
   
  
**Root Layout (Required)**

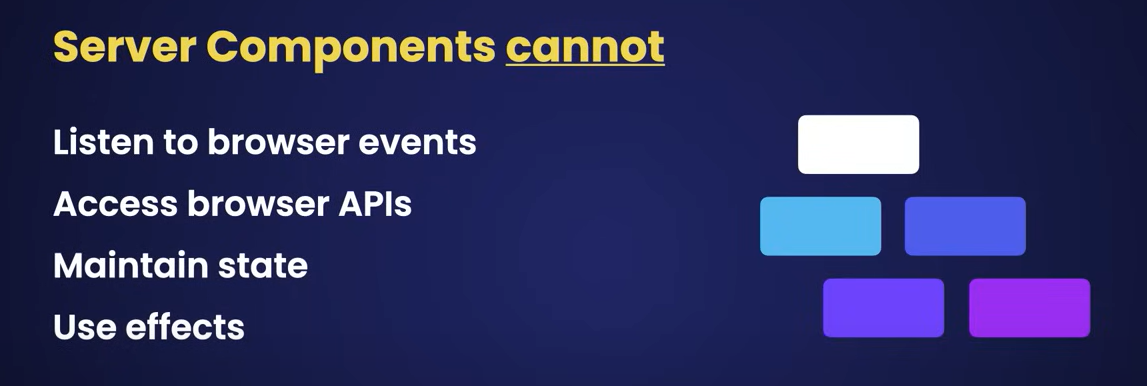
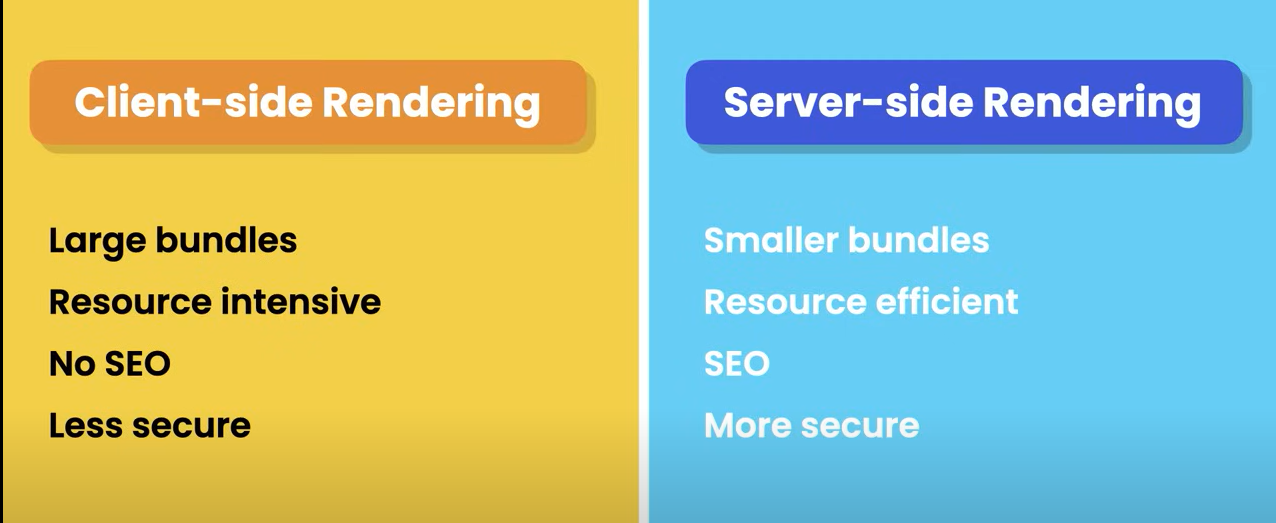
The root layout is defined at the top level of the app directory and **applies to all routes**. This layout is required and **must contain html and body tags**, allowing you to modify the initial HTML returned from the server.

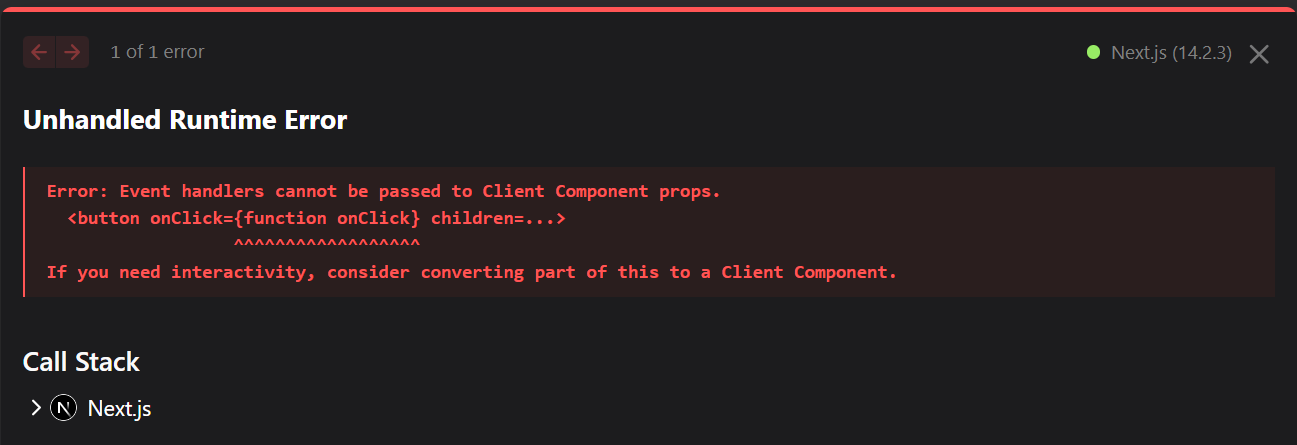
   
 **Root Layout ( App/layout.tsx )** **Multiple Layout example  
  
Pages**

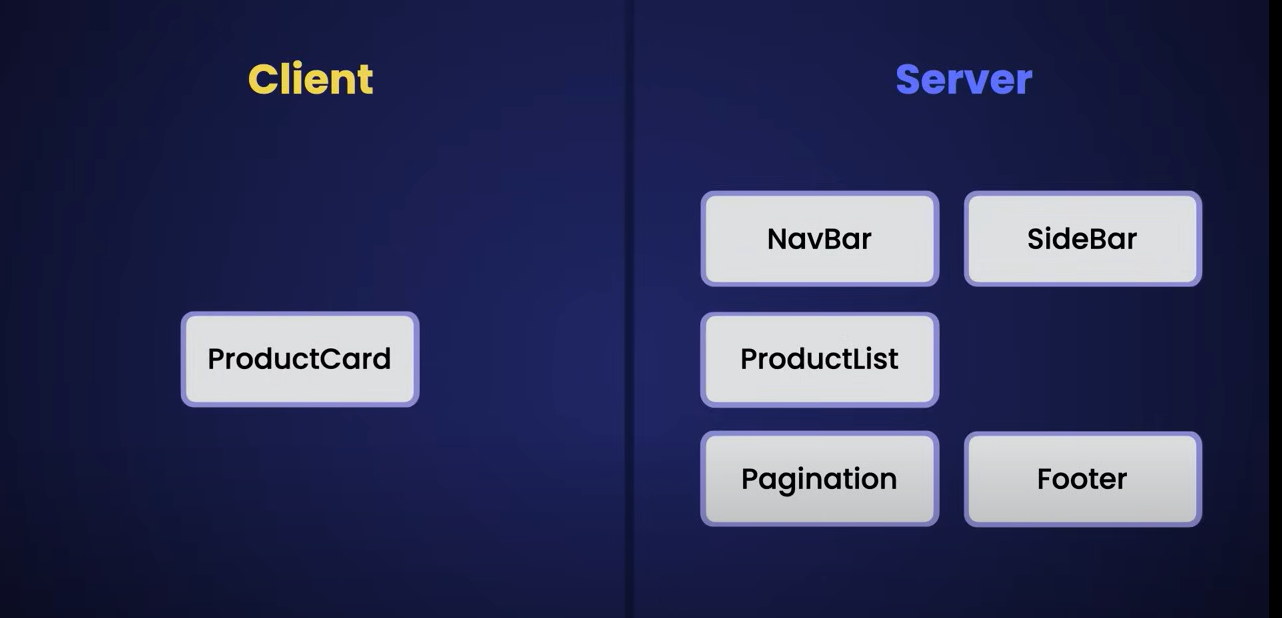
A page is UI that is unique to a route. You can define a page by default exporting a component from a page.js file.

For example, to create your index page, add the page.js file inside the app directory:

**ROUING AND NAVIGATION:-**



To use the USER INTERACTION on Client side :-  
🡪 We can convert server component to client component by using **‘use client’;** directive on top line of any component.  
🡪 otherwise the best approach would be to extract only the interactive element and make it the client component .   
**example:-** separate the button element and make it a different client side component and then importing it in the card component

**DATA FETCHING**

**CACHING:-**Technique to store data somewhere that is faster to access (i.e. Memory, File System, Network, **Note**:- fetching data while going from memory to network is always slower)

Hence Next js has build in data cache , that is lets say we fetch some data with a URL xyz  
then after data being fetched Next js save it in its Data Cache which is based on the File System.  
 So if the same URL got called again to fetch data then Next js would return it from its Data cache or File system.

Note:- Now if we have data that changes frequently, we can disable data to store in cache or treat this data as fresh data always using **‘no-store’** (which would disable caching)**.**

const res = await fetch(

'https://jsonplaceholder.typicode.com/users',

{

cache: 'no-store'

});  
Other option to keep data fresh for a certain period of time using in-built Next js config object.

const res = await fetch(

'https://jsonplaceholder.typicode.com/users',

{

next: {revalidate:10} // revalidate after every 10 seconds

});

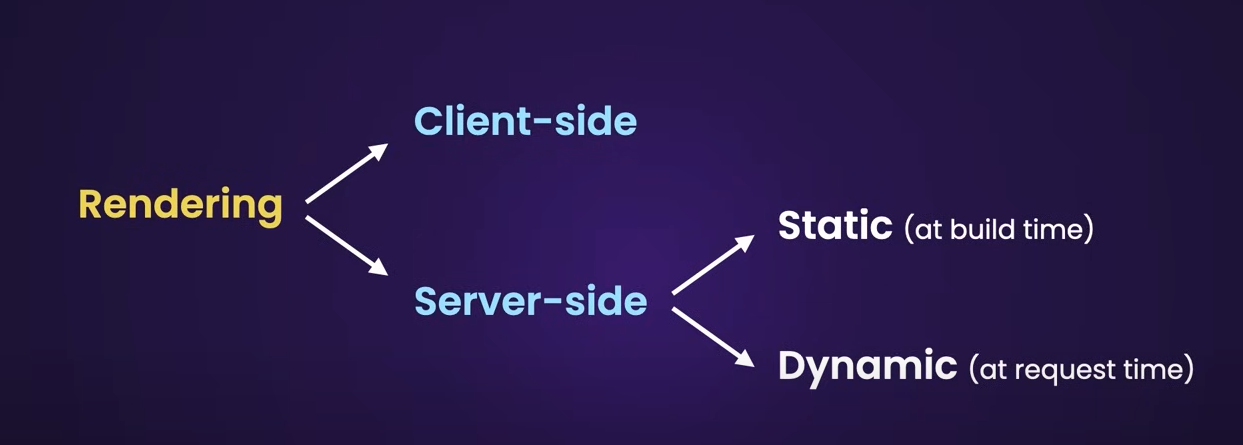
[Note:- Caching behaviour **can only be** implemented in Fetch function]

**STATIC AND DYNAMIC RENDERING:-**

**Static rendering:-**In Next JS we have another performance optimization technique, that is **Static Rendering** or **Static Site Generation** (Rendering at build time).  
Basically, if we have any page with static data, we can have Next JS render them once when we build our application for production. So next time those pages or component are needed Next JS not going to re-render. It’s going to get them from payload or cache which is based on the File system.

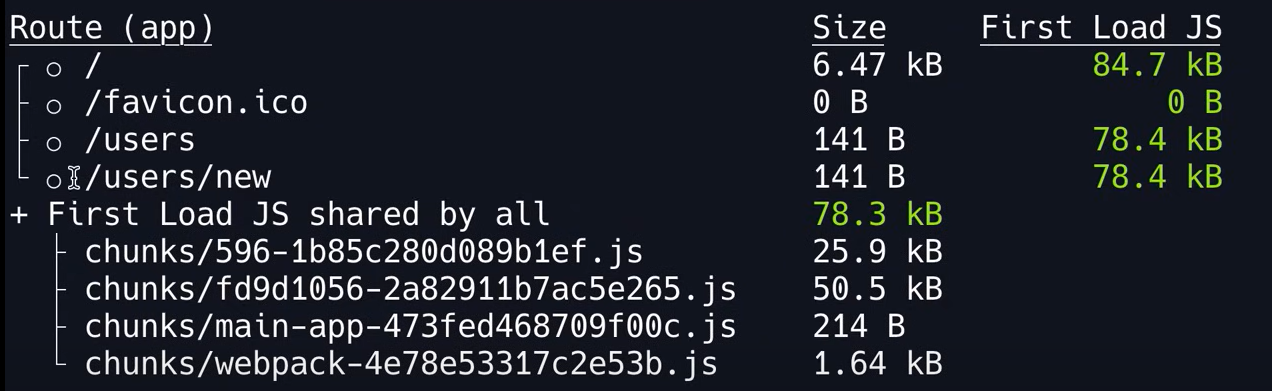
**Dynamic Rendering:-**

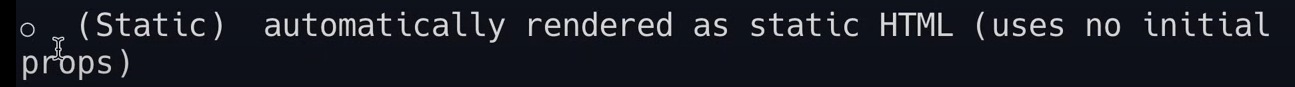
Rendering happens at request time.



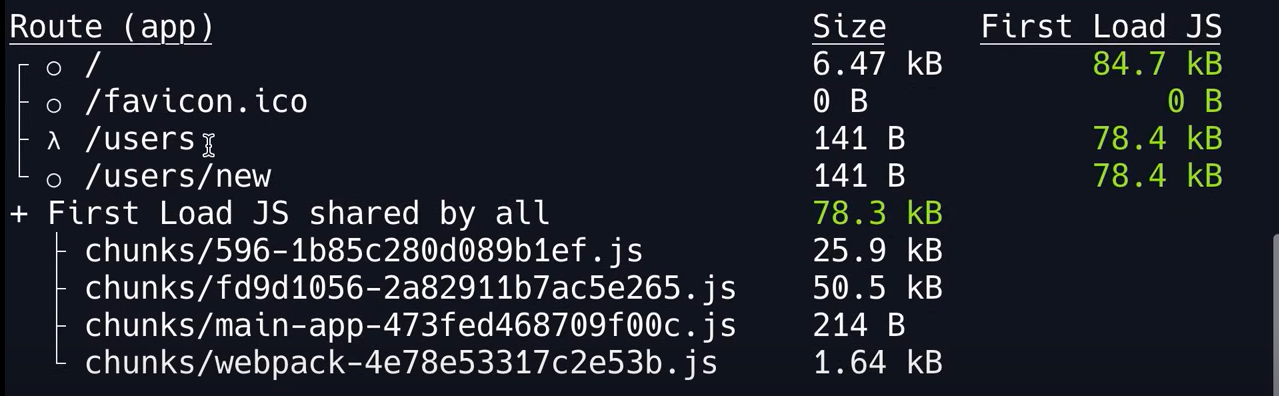
In production build if we don’t mention any type of cache, Next js will think it’s a STATIC PAGE and to make it DYNAMIC we can implement **cache: 'no-store'** or **next: {revalidate:10}**You can check by npm run build >> npm start ( prod) with or without above cache

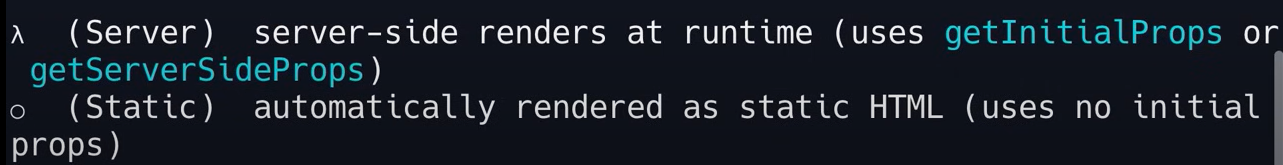
**STATIC BUILD**



* 

**DYNAMIC BUILD**



* 

**STYLING:-**

Next.js supports different ways of styling your application, including:

**Global CSS**: Simple to use and familiar for those experienced with traditional CSS, but can lead to larger CSS bundles and difficulty managing styles as the application grows.

**CSS Modules**: Create locally scoped CSS classes to avoid naming conflicts and improve maintainability.

\**Doesn’t support card-container, it has to be cardContainer*

**Tailwind CSS**: A utility-first CSS framework that allows for rapid custom designs by composing utility classes.

**Sass**: A popular CSS preprocessor that extends CSS with features like variables, nested rules, and mixins.

**CSS-in-JS**: Embed CSS directly in your JavaScript components, enabling dynamic and scoped styling.